

Tema 8: Técnicas de control y diseño modular

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo
Francisco J. Martín Mateos

Dpto. de Ciencias de la Computación e Inteligencia Artificial

UNIVERSIDAD DE SEVILLA

Nim con fases

● Sesión

```
CLIPS> (load "nim.clp")
CLIPS> $*****$*
TRUE
CLIPS> (reset)
CLIPS> (run)
Elige quien empieza: computadora o Humano (c/h) c
Escribe el numero de piezas: 15
La computadora coge 2 pieza(s)
Quedan 13 pieza(s)
Escribe el numero de piezas que coges: 3
Quedan 10 pieza(s)
La computadora coge 1 pieza(s)
Quedan 9 pieza(s)
Escribe el numero de piezas que coges: 2
Quedan 7 pieza(s)
La computadora coge 2 pieza(s)
Quedan 5 pieza(s)
Escribe el numero de piezas que coges: 4
Tiene que elegir un numero entre 1 y 3
Escribe el numero de piezas que coges: 1
Quedan 4 pieza(s)
La computadora coge 3 pieza(s)
Quedan 1 pieza(s)
Tienes que coger la ultima pieza
Has perdido
CLIPS>
```

Nim con fases

- Elección del jugador

```
(deffacts fase-inicial
  (fase elige-jugador))
```

```
(defrule elige-jugador
  (fase elige-jugador)
  =>
  (printout t "Elige quien empieza: ")
  (printout t "computadora o Humano (c/h) ")
  (assert (jugador-elegido (read))))
```

```
(defrule correcta-eleccion-de-jugador
  ?fase <- (fase elige-jugador)
  ?eleccion <- (jugador-elegido ?jugador&c|h)
  =>
  (retract ?fase ?eleccion)
  (assert (turno ?jugador))
  (assert (fase elige-numero-de-piezas)))
```

```
(defrule incorrecta-eleccion-de-jugador
  ?fase <- (fase elige-jugador)
  ?eleccion <- (jugador-elegido ?jugador&~c&~h)
  =>
  (retract ?fase ?eleccion)
  (printout t ?jugador " es distinto de c y h" crlf)
  (assert (fase elige-jugador)))
```

Nim con fases

- Elección del número de piezas

```
(defrule elige-numero-de-piezas
  (fase elige-numero-de-piezas)
  =>
  (printout t "Escribe el numero de piezas: ")
  (assert (numero-de-piezas (read))))

(defrule correcta-eleccion-del-numero-de-piezas
  ?fase <- (fase elige-numero-de-piezas)
  ?eleccion <- (numero-de-piezas ?n&:(integerp ?n)
               &:(> ?n 0))
  =>
  (retract ?fase ?eleccion)
  (assert (numero-de-piezas ?n)))

(defrule incorrecta-eleccion-del-numero-de-piezas
  ?fase <- (fase elige-numero-de-piezas)
  ?eleccion <- (numero-de-piezas ?n&~:(integerp ?n)
               |:(<= ?n 0))
  =>
  (retract ?fase ?eleccion)
  (printout t ?n " no es un numero entero mayor que 0"
           crlf)
  (assert (fase elige-numero-de-piezas)))
```

Nim con fases

- Jugada humana

```
(defrule pierde-el-humano
  (turno h)
  (numero-de-piezas 1)
  =>
  (printout t "Tienes que coger la ultima pieza" crlf)
  (printout t "Has perdido" crlf))
```

```
(defrule eleccion-humana
  (turno h)
  (numero-de-piezas ?n&:(> ?n 1))
  =>
  (printout t "Escribe el numero de piezas que coges: ")
  (assert (piezas-cogidas (read))))
```

Nim con fases

```
(defrule correcta-eleccion-humana
  ?pila <- (numero-de-piezas ?n)
  ?eleccion <- (piezas-cogidas ?m)
  ?turno <- (turno h)
  (test (and (integerp ?m)
             (>= ?m 1)
             (<= ?m 3)
             (< ?m ?n)))

=>
  (retract ?pila ?eleccion ?turno)
  (bind ?nuevo-numero-de-piezas (- ?n ?m))
  (assert (numero-de-piezas ?nuevo-numero-de-piezas))
  (printout t "Quedan "
            ?nuevo-numero-de-piezas " pieza(s)" crlf)
  (assert (turno c)))

(defrule incorrecta-eleccion-humana
  (numero-de-piezas ?n)
  ?eleccion <- (piezas-cogidas ?m)
  ?turno <- (turno h)
  (test (or (not (integerp ?m))
            (< ?m 1)
            (> ?m 3)
            (>= ?m ?n)))

=>
  (retract ?eleccion ?turno)
  (printout t "Tiene que elegir un numero entre 1 y 3"
            crlf)
  (assert (turno h)))
```

Nim con fases

- Jugada de la computadora

```
(defrule pierde-la-computadora
  (turno c)
  (numero-de-piezas 1)
  =>
  (printout t "La computadora coge la ultima pieza"
            crlf "He perdido" crlf))

(deffacts heuristica
  (computadora-coge 1 cuando-el-resto-es 1)
  (computadora-coge 1 cuando-el-resto-es 2)
  (computadora-coge 2 cuando-el-resto-es 3)
  (computadora-coge 3 cuando-el-resto-es 0))

(defrule eleccion-computadora
  ?turno <- (turno c)
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  (computadora-coge ?m cuando-el-resto-es =(mod ?n 4))
  =>
  (retract ?turno ?pila)
  (printout t "La computadora coge " ?m " pieza(s)"
            crlf)
  (bind ?nuevo-numero-de-piezas (- ?n ?m))
  (printout t "Quedan " ?nuevo-numero-de-piezas
            " pieza(s)" crlf)
  (assert (numero-de-piezas ?nuevo-numero-de-piezas))
  (assert (turno h)))
```

Control empotrado en las reglas

- **Fases en el Nim:**
 - Elección del jugador.
 - Elección del número de piezas.
 - Turno del humano.
 - Turno de la computadora.
- **Hechos de control:**
 - (fase elige-jugador)
 - (fase elige-numero-de-piezas)
 - (turno h)
 - (turno c)
- **Inconvenientes del control empotrado en las reglas:**
 - Dificultad para entenderse.
 - Dificultad para precisar la conclusión de cada fase.

Técnicas de control

- Ejemplo de fases de un problema:
 - Detección.
 - Aislamiento.
 - Recuperación.
- Técnicas de control:
 - Control empotrado en las reglas.
 - Prioridades.
 - Reglas de control.
 - Módulos.

Prioridades

- **Ejemplo:** ej-1.clp

```
(deffacts inicio
  (prioridad primera)
  (prioridad segunda)
  (prioridad tercera))
```

```
(defrule regla-1
  (prioridad primera)
  =>
  (printout t "Escribe primera" crlf))
```

```
(defrule regla-2
  (prioridad segunda)
  =>
  (printout t "Escribe segunda" crlf))
```

```
(defrule regla-3
  (prioridad tercera)
  =>
  (printout t "Escribe tercera" crlf))
```

Prioridades

- Sesión

```
CLISP> (load "ej-1.clp")
CLIPS> $***
TRUE
CLIPS> (reset)
CLIPS> (facts)
f-0      (initial-fact)
f-1      (prioridad primera)
f-2      (prioridad segunda)
f-3      (prioridad tercera)
For a total of 4 facts.
CLIPS> (rules)
regla-1
regla-2
regla-3
For a total of 3 defrules.
CLIPS> (agenda)
0      regla-3: f-3
0      regla-2: f-2
0      regla-1: f-1
For a total of 3 activations.
CLIPS> (run)
Escribe tercera
Escribe segunda
Escribe primera
CLIPS>
```

Prioridades

- Ejemplo: ej-2.clp

```
(deffacts inicio
  (prioridad primera)
  (prioridad segunda)
  (prioridad tercera))
```

```
(defrule regla-1
  (declare (salience 30))
  (prioridad primera)
  =>
  (printout t "Escribe primera" crlf))
```

```
(defrule regla-2
  (declare (salience 20))
  (prioridad segunda)
  =>
  (printout t "Escribe segunda" crlf))
```

```
(defrule regla-3
  (declare (salience 10))
  (prioridad tercera)
  =>
  (printout t "Escribe tercera" crlf))
```

Prioridades

- Sesión

```
CLIPS> (load "ej-2.clp")
CLIPS> $***
TRUE
CLIPS> (reset)
CLIPS> (facts)
f-0      (initial-fact)
f-1      (prioridad primera)
f-2      (prioridad segunda)
f-3      (prioridad tercera)
For a total of 4 facts.
CLIPS> (rules)
regla-1
regla-2
regla-3
For a total of 3 defrules.
CLIPS> (agenda)
30      regla-1: f-1
20      regla-2: f-2
10      regla-3: f-3
For a total of 3 activations.
CLIPS> (run)
Escribe primera
Escribe segunda
Escribe tercera
CLIPS>
```

Prioridades

- **Sintaxis:**
 - (declare (saliencia <numero>))
- **Valores:**
 - Mínimo: -10000
 - Máximo: 10000
 - Defecto: 0
- **Inconvenientes:**
 - Abuso.
 - Contradicción con el objetivo de los sistemas basados en reglas.

Reglas de control

- Reglas de cambio de fases ej-3.clp

```
(defrule deteccion-a-aislamiento
  (declare (salience -10))
  ?fase <- (fase deteccion)
  =>
  (retract ?fase)
  (assert (fase aislamiento)))
```

```
(defrule aislamiento-a-recuperacion
  (declare (salience -10))
  ?fase <- (fase aislamiento)
  =>
  (retract ?fase)
  (assert (fase recuperacion)))
```

```
(defrule recuperacion-a-deteccion
  (declare (salience -10))
  ?fase <- (fase recuperacion)
  =>
  (retract ?fase)
  (assert (fase deteccion)))
```

```
(defrule detecta-fuego
  (fase deteccion)
  (luz-a roja)
  =>
  (assert (problema fuego)))
```

```
(defacts inicio
  (fase deteccion) (luz-a roja))
```

Reglas de control

- Sesión

```
CLIPS> (load "ej-3.clp")
CLIPS> ****$
TRUE
CLIPS> (watch rules)
CLIPS> (reset)
CLIPS> (run 7)
FIRE      1 detecta-fuego: f-1,f-2
FIRE      2 deteccion-a-aislamiento: f-1
FIRE      3 aislamiento-a-recuperacion: f-4
FIRE      4 recuperacion-a-deteccion: f-5
FIRE      5 detecta-fuego: f-6,f-2
FIRE      6 deteccion-a-aislamiento: f-6
FIRE      7 aislamiento-a-recuperacion: f-7
CLIPS>
```


Reglas de control

- Cambio de fase mediante una regla ej-4.clp

```
(deffacts control
  (fase deteccion)
  (siguiente-fase deteccion aislamiento)
  (siguiente-fase aislamiento recuperacion)
  (siguiente-fase recuperacion deteccion))
```

```
(defrule cambio-de-fase
  (declare (salience -10))
  ?fase <- (fase ?actual)
  (siguiente-fase ?actual ?siguiente)
  =>
  (retract ?fase)
  (assert (fase ?siguiente)))
```

```
(defrule detecta-fuego
  (fase deteccion)
  (luz-a roja)
  =>
  (assert (problema fuego)))
```

```
(deffacts inicio
  (fase deteccion)
  (luz-a roja))
```

Reglas de control

```
CLIPS> (load "ej-4.clp")
$**$
TRUE
CLIPS> (watch rules)
CLIPS> (watch facts)
CLIPS> (reset)
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (fase deteccion)
==> f-2      (siguiente-fase deteccion aislamiento)
==> f-3      (siguiente-fase aislamiento recuperacion)
==> f-4      (siguiente-fase recuperacion deteccion)
==> f-5      (luz-a roja)
CLIPS> (run 5)
FIRE      1 detecta-fuego: f-1,f-5
==> f-6      (problema fuego)
FIRE      2 cambio-de-fase: f-1,f-2
<== f-1      (fase deteccion)
==> f-7      (fase aislamiento)
FIRE      3 cambio-de-fase: f-7,f-3
<== f-7      (fase aislamiento)
==> f-8      (fase recuperacion)
FIRE      4 cambio-de-fase: f-8,f-4
<== f-8      (fase recuperacion)
==> f-9      (fase deteccion)
FIRE      5 detecta-fuego: f-9,f-5
```

Reglas de control

- Cambio de fase mediante una regla y sucesión de fases ej-5.clp

```
(deffacts control
  (fase deteccion)
  (sucesion-de-fases aislamiento
                        recuperacion
                        deteccion))
```

```
(defrule cambio-de-fase
  (declare (salience -10))
  ?fase <- (fase ?actual)
  (sucesion-de-fases ?siguiente $?resto)
  =>
  (retract ?fase)
  (assert (fase ?siguiente))
  (assert (sucesion-de-fases ?resto ?siguiente)))
```

Reglas de control

```
CLIPS> (clear)
CLIPS> (load "ej-5.clp")
$**$
TRUE
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (reset)
==> f-0      (initial-fact)
==> f-1      (fase deteccion)
==> f-2      (sucesion-de-fases
              aislamiento recuperacion deteccion)
==> f-3      (luz-a roja)
CLIPS> (run 5)
FIRE      1 detecta-fuego: f-1,f-3
==> f-4      (problema fuego)
FIRE      2 cambio-de-fase: f-1,f-2
<== f-1      (fase deteccion)
==> f-5      (fase aislamiento)
==> f-6      (sucesion-de-fases
              recuperacion deteccion aislamiento)
FIRE      3 cambio-de-fase: f-5,f-6
<== f-5      (fase aislamiento)
==> f-7      (fase recuperacion)
==> f-8      (sucesion-de-fases
              deteccion aislamiento recuperacion)
FIRE      4 cambio-de-fase: f-7,f-8
<== f-7      (fase recuperacion)
==> f-9      (fase deteccion)
FIRE      5 detecta-fuego: f-9,f-3
```

Nim con módulos

- **Módulo MAIN**

```
(defmodule MAIN
  (export deftemplate numero-de-piezas
          turno
          initial-fact))
```

```
(defrule MAIN::inicio
  =>
  (focus INICIO))
```

```
(defrule MAIN::la-computadora-elige
  ?turno <- (turno c)
  (numero-de-piezas ~0)
  =>
  (focus COMPUTADORA)
  (retract ?turno)
  (assert (turno h)))
```

```
(defrule MAIN::el-humano-elige
  ?turno <- (turno h)
  (numero-de-piezas ~0)
  =>
  (focus HUMANO)
  (retract ?turno)
  (assert (turno c)))
```

Nim con módulos

- **Módulo INICIO**

```
(defmodule INICIO
  (import MAIN deftemplate numero-de-piezas
            turno
            initial-fact))

(defrule INICIO::elige-jugador
  (not (turno ?))
  =>
  (printout t "Elige quien empieza: "
            "computadora o Humano (c/h) ")
  (assert (turno (read))))

(defrule INICIO::incorrecta-eleccion-de-jugador
  ?eleccion <- (turno ?jugador&~c&~h)
  =>
  (retract ?eleccion)
  (printout t ?jugador " es distinto de c y h" crlf))

(defrule INICIO::elige-numero-de-piezas
  (not (numero-de-piezas-elegidas ?))
  =>
  (printout t "Escribe el numero de piezas: ")
  (assert (numero-de-piezas-elegidas (read))))
```

Nim con módulos

```
(defrule INICIO::incorrecta-eleccion-del-numero-de-piezas
  ?e <- (numero-de-piezas-elegidas ?n&~:(integerp ?n)
        |:(<= ?n 0))

=>
(retract ?e)
(printout t ?n " no es un numero entero mayor que 0"
         crlf))

(defrule INICIO::correcta-eleccion-del-numero-de-piezas
  (numero-de-piezas-elegidas ?n&:(integerp ?n)
   &:(> ?n 0))

=>
(assert (numero-de-piezas ?n))
(return))
```

● Módulo HUMANO

```
(defmodule HUMANO
  (import MAIN deftemplate numero-de-piezas))

(defrule HUMANO::pierde-el-humano
  ?h <- (numero-de-piezas 1)

=>
(printout t "Tienes que coger la ultima pieza" crlf)
(printout t "Has perdido" crlf)
(retract ?h)
(assert (numero-de-piezas 0)))
```

Nim con módulos

```
(defrule HUMANO::eleccion-humana
  (numero-de-piezas ?n&:(> ?n 1))
  (not (piezas-cogidas ?))
  =>
  (printout t "Escribe el numero de piezas que coges: ")
  (assert (piezas-cogidas (read))))
```

```
(defrule HUMANO::correcta-eleccion-humana
  ?pila <- (numero-de-piezas ?n)
  ?eleccion <- (piezas-cogidas ?m)
  (test (and (integerp ?m)
             (>= ?m 1)
             (<= ?m 3)
             (< ?m ?n)))
  =>
  (retract ?pila ?eleccion)
  (bind ?nuevo-numero-de-piezas (- ?n ?m))
  (assert (numero-de-piezas ?nuevo-numero-de-piezas))
  (printout t "Quedan " ?nuevo-numero-de-piezas
            " pieza(s)" crlf)
  (return))
```

```
(defrule HUMANO::incorrecta-eleccion-humana
  (numero-de-piezas ?n)
  ?eleccion <- (piezas-cogidas ?m)
  (test (or (not (integerp ?m))
            (< ?m 1)
            (> ?m 3)
            (>= ?m ?n)))
  =>
  (printout t "Tiene que elegir un numero entre 1 y 3"
            crlf)
  (retract ?eleccion))
```


Nim con módulos

● Módulo COMPUTADORA

```
(defmodule COMPUTADORA
  (import MAIN deftemplate numero-de-piezas))

(defrule COMPUTADORA::pierde-la-computadora
  ?h <- (numero-de-piezas 1)
  =>
  (printout t "La computadora coge la ultima pieza" crlf)
  (printout t "He perdido" crlf)
  (retract ?h)
  (assert (numero-de-piezas 0)))

(deffacts heuristica
  (computadora-coge 1 cuando-el-resto-es 1)
  (computadora-coge 1 cuando-el-resto-es 2)
  (computadora-coge 2 cuando-el-resto-es 3)
  (computadora-coge 3 cuando-el-resto-es 0))

(defrule COMPUTADORA::eleccion-computadora
  ?pila <- (numero-de-piezas ?n&:(> ?n 1))
  (computadora-coge ?m cuando-el-resto-es =(mod ?n 4))
  =>
  (retract ?pila)
  (printout t "La computadora coge " ?m " pieza(s)" crlf)
  (bind ?nuevo-numero-de-piezas (- ?n ?m))
  (printout t "Quedan " ?nuevo-numero-de-piezas
    " pieza(s)" crlf)
  (assert (numero-de-piezas ?nuevo-numero-de-piezas))
  (return))
```

Nim con módulos

- Sesión con traza:

```
CLIPS> (watch facts)
CLIPS> (watch rules)
CLIPS> (watch focus)
CLIPS> (reset)
==> Focus MAIN
==> f-0      (initial-fact)
==> f-1      (computadora-coge 1 cuando-el-resto-es 1)
==> f-2      (computadora-coge 1 cuando-el-resto-es 2)
==> f-3      (computadora-coge 2 cuando-el-resto-es 3)
==> f-4      (computadora-coge 3 cuando-el-resto-es 0)
CLIPS> (run)
FIRE      1 inicio: f-0
==> Focus INICIO from MAIN
FIRE      2 elige-jugador: f-0,
Elige quien empieza: computadora o Humano (c/h) a
==> f-5      (turno a)
FIRE      3 incorrecta-eleccion-de-jugador: f-5
<== f-5      (turno a)
a es distinto de c y h
FIRE      4 elige-jugador: f-0,
Elige quien empieza: computadora o Humano (c/h) c
==> f-6      (turno c)
FIRE      5 elige-numero-de-piezas: f-0,
Escribe el numero de piezas: a
==> f-7      (numero-de-piezas-elegidas a)
FIRE      6 incorrecta-eleccion-del-numero-de-piezas: f-7
<== f-7      (numero-de-piezas-elegidas a)
a no es un numero entero mayor que 0
```

Nim con módulos

```
FIRE    7 elige-numero-de-piezas: f-0,
Escribe el numero de piezas: 5
==> f-8      (numero-de-piezas-elegidas 5)
FIRE    8 correcta-eleccion-del-numero-de-piezas: f-8
==> f-9      (numero-de-piezas 5)
<== Focus INICIO to MAIN
FIRE    9 la-computadora-elige: f-6,f-9
==> Focus COMPUTADORA from MAIN
<== f-6      (turno c)
==> f-10     (turno h)
FIRE   10 eleccion-computadora: f-9,f-1
<== f-9      (numero-de-piezas 5)
La computadora coge 1 pieza(s)
Quedan 4 pieza(s)
==> f-11     (numero-de-piezas 4)
<== Focus COMPUTADORA to MAIN
FIRE   11 el-humano-elige: f-10,f-11
==> Focus HUMANO from MAIN
<== f-10     (turno h)
==> f-12     (turno c)
FIRE   12 eleccion-humana: f-11,
Escribe el numero de piezas que coges: 4
==> f-13     (piezas-cogidas 4)
FIRE   13 incorrecta-eleccion-humana: f-11,f-13
Tiene que elegir un numero entre 1 y 3
<== f-13     (piezas-cogidas 4)
FIRE   14 eleccion-humana: f-11,
Escribe el numero de piezas que coges: 1
==> f-14     (piezas-cogidas 1)
```

Nim con módulos

```
FIRE 15 correcta-eleccion-humana: f-11,f-14
<== f-11 (numero-de-piezas 4)
<== f-14 (piezas-cogidas 1)
==> f-15 (numero-de-piezas 3)
Quedan 3 pieza(s)
<== Focus HUMANO to MAIN
FIRE 16 la-computadora-elige: f-12,f-15
==> Focus COMPUTADORA from MAIN
<== f-12 (turno c)
==> f-16 (turno h)
FIRE 17 eleccion-computadora: f-15,f-3
<== f-15 (numero-de-piezas 3)
La computadora coge 2 pieza(s)
Quedan 1 pieza(s)
==> f-17 (numero-de-piezas 1)
<== Focus COMPUTADORA to MAIN
FIRE 18 el-humano-elige: f-16,f-17
==> Focus HUMANO from MAIN
<== f-16 (turno h)
==> f-18 (turno c)
FIRE 19 pierde-el-humano: f-17
Tienes que coger la ultima pieza
Has perdido
<== f-17 (numero-de-piezas 1)
==> f-19 (numero-de-piezas 0)
<== Focus HUMANO to MAIN
<== Focus MAIN
```